DOCUMENT RESUME

ED 388 251 IR 017 400

AUTHOR Harris, Julian; Maurer, Hermann
TITLE HyperCard Monitor System.
PUB DATE 94
NOTE 6p.; In: Educational Multimedia and Hypermedia, 1994. Proceedings of ED-MEDIA 94--World Conference on Educational Multimedia and Hypermedia (Vancouver, British Columbia, Canada, June 25-30, 1994); see IR 017 359.
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computer Software; Foreign Countries; Multimedia Materials; *Statistical Analysis; Statistical Data; *Use Studies
IDENTIFIERS Computer Users; *HyperCard; *Monitoring

ABSTRACT

An investigation into high level event monitoring within the scope of a well-known multimedia application, HyperCard--a program on the Macintosh computer, is carried out. A monitoring system is defined as a system which automatically monitors usage of some activity and gathers statistics based on what is has observed. Monitor systems can give the authors of a program or system accurate statistics on how their product is being used. An attempt is made to gather and process user's actions at a high level, enabling accurate and quantitative statistical analysis to be performed. This type of information is either very difficult or impossible to obtain through other methods. The HyperCard monitor works by inserting itself in the HyperCard message-passing hierarchy and setting up handlers for all messages considered useful for data gathering. Monitor data can be used to analyze a particular user for a particular work session, or to offer a head to head comparison of users over many sessions. Overall, the HyperCard Monitor System was found to be very effective, and many of the general concepts used for the system could be extended to the operating system (OS) level if such an OS provided a facility for generating and interpreting high level events of this nature. Performance on a Macintosh Quadra 700 was very good and the resulting data output was very compact--tests revealed about 50k of data per user per half hour. Contains six references. (MAS)

HyperCard Monitor System

JULIAN HARRIS, HERMANN MAURER
Hypermedia Unit
Computer Science Department
The University of Auckland, Private Bag, Auckland
Email: J_Harris@cs.aukuni.ac.nz, H_Maurer@cs.auckland.ac.nz

Abstract: An investigation into high level event monitoring within the scope of a well-known multimedia application called HyperCard is carried out. An attempt is made to gather and process user's actions at a high level, enabling accurate and quantitative statistical analysis to be performed. This type of information is either very difficult or impossible to obtain through other methods.

Introduction

In the computer world today, monitor systems are a reasonably recent development. For the purposes of this paper, we shall define a monitor to be a system which automatically monitors usage of some activity and gathers statistics based on what it has observed. Monitor systems are useful tools in many areas. For instance public computer-based kiosk systems have been used which gather information on what the user did while using the system (Maurer, Sammer & Schneider 1993). The system they employed was automatic but only involved audio and video recordings of what users did, thus making analysis a tedious manual process. Monitor systems can give the authors of a program or system that is being monitored very accurate statistics on *how their product is being used.* From the resulting information, they can assess what aspects of a product need the most attention.

One reason monitor systems are rare is the ratio between the scope of the monitor (what sort of information it collects) and the type of information. The more you limit the scope of a monitor system (eg. a system to monitor activity for a particular application) the easier it becomes to record useable information. However, the usefulness of such a specialised analysis tool is limited and in most cases infeasible. The most useful monitor system would be integrated with the Operating System (OS) itself, so that activity of every program running under the OS can be monitored. The main problem with this is that the only information the monitor system would obtain is information that is available to the system, which is usually very low level and of little general analytical value.

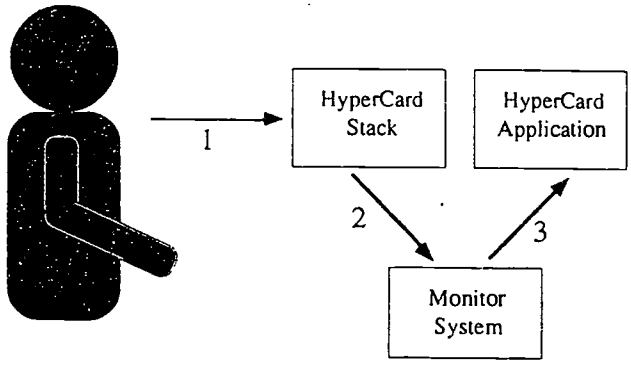
Currently the only way to find out how people are using a system is to have other people, cameras, or microphones observe the users, possibly having to record the data manually. This is problematic because of several reasons. One is that the mere fact that the observers are there will influence how the users behave, introducing unwanted biases. Another point is that this method is very costly, time consuming and accurate results can be very difficult to obtain. This is usually because of the distance between the data generator (the user) and the data gatherer (the observer). For example, a common approach is to ask the user at the end of the session to fill out a questionnaire or put down any comments they have about it. This approach will only yield general information, and will tend to be fairly inaccurate. A suitably designed computer monitor system will be able to retrieve the data *directly* from the source which will vastly improve the integrity of the data and will be able to catch much more useful information as well.

On the Macintosh Computer, there is a program called HyperCard which is a simple, easy to use multimedia system (Apple HyperCard, 1988). There is an abundance of programs (called stacks) created using HyperCard. The aim of this paper was to develop a simple HyperCard-based usage monitor which monitors Macintosh users as they run HyperCard stacks, and stores the resulting data away for later statistical analysis. In doing this, the scope of the monitor system is being limited to user activity that goes on within the HyperCard application while enabling us to easily catch high level information.

Overall Structure

A HyperCard stack is made up of objects arranged in a hierarchy. A stack owns a collection of backgrounds. Backgrounds own a collection of cards, and cards own a collection of fields and buttons.

When a user does anything in HyperCard, messages are generated which are sent to these objects through a message passing hierarchy (for instance, clicking somewhere generates a mouseup message). Each object has a script, within which can be a series of message handlers which are executed when the appropriate message is sent to that object. Message handlers consist of a series of English-like instructions (HyperTalk) like "Get the first field / if it is empty then answer 'That field is empty.'"



1 User Creates a HyperCard message (via mouse or keyboard or other input device)
2 Once the stack has handled it, it is passed on explicitly to the HyperCard Monitor
3 Whether or not the message is trapped by the Monitor, the event is then passed on to the HyperCard Application

Figure 1. How messages are caught by the HyperCard Monitor System

In this message handler, the message can be passed on further down the hierarchy for further processing. Handlers have to explicitly pass the message on if they want other objects further down the message hierarchy to get the message — it is considered good practise to pass the standard HyperCard messages on for other objects.

Our HyperCard monitor works by inserting itself in the HyperCard message-passing hierarchy (Figure 1) and sets up handlers for all the messages that are considered useful for data gathering. This requires that all messages the monitor handles are passed explicitly by the object that got the messages in the first place.

As many stacks do not actually pass these messages in all of their handlers, one thing that had to be done was create a utility which would modify a given stack to ensure that it would pass enough messages for our monitor to work since the monitor expects events to happen in a certain order (see Figure 2) .

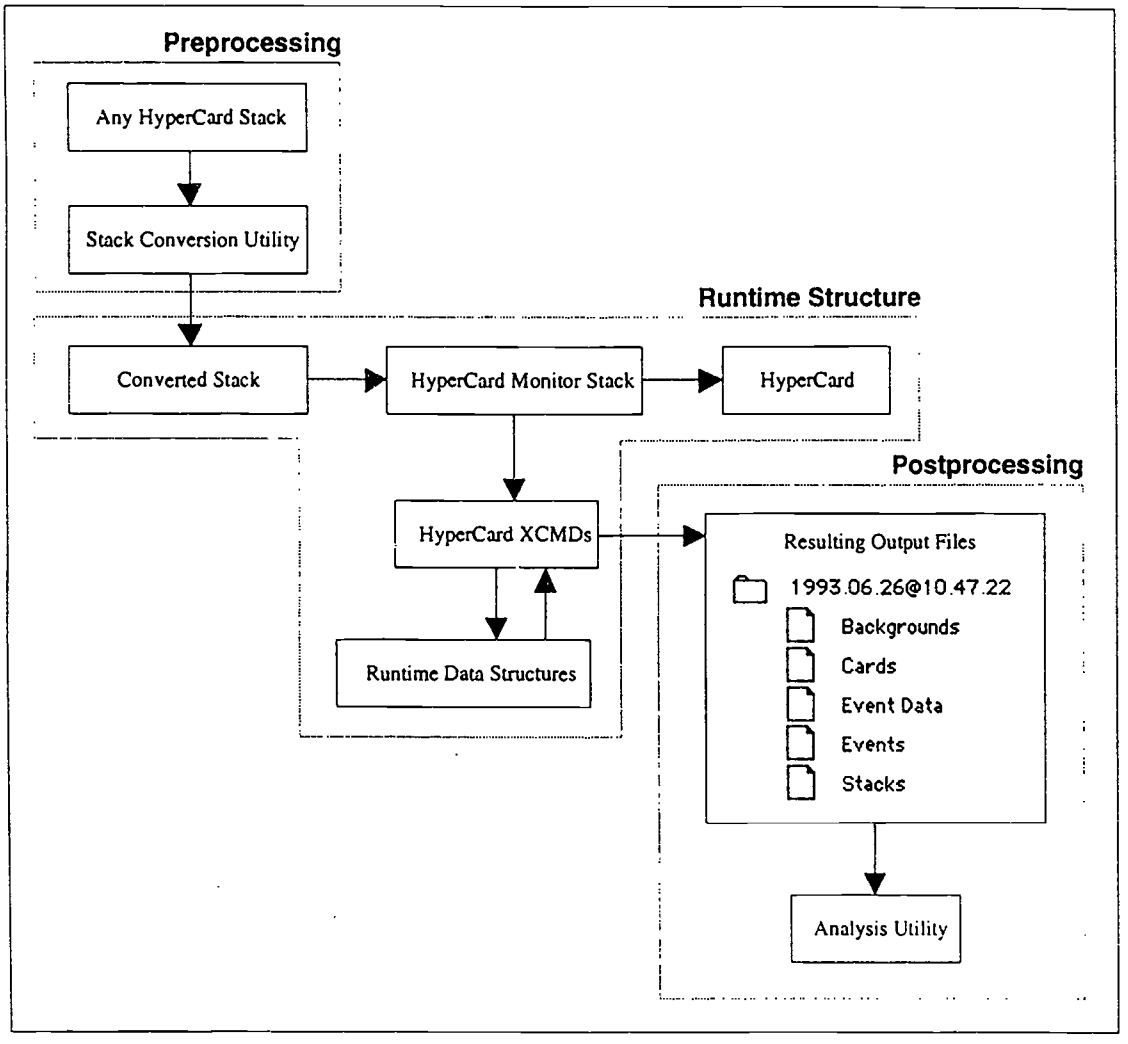


Figure 2. An overall view of the HyperCard Monitor System Structure

The Monitor System

There are many levels of statistical information that can be derived from the monitor data. One is analysis of a particular user for a particular work session. Another is head to head comparison of users over many sessions which involves more general statistical analysis. In systems like the proposed University Transaction, Information and Communication System (UTICS) (Schneider, 1993) which use fundamentally new interfaces between computers and humans, both head-to-head comparisons and overall statistical information is useful to assist in research when seeking to refine or remodel the interface.

Whatever data is analysed, there need to be clearly defined points which say when a user stops using the system and another user starts. This is a general problem and usually schemes like finishing a session after a predetermined period of inactivity are employed.

We defined the start of a session to be when the user starts up HyperCard (suitably modified so that it launches our monitor), when the user opens the monitor directly, opens a stack from the home stack, or finally when the user explicitly chooses to start a session using the monitor configuration. Most of the time it is usually one of the first two. Finishing a session is defined to occur when a user closes the monitor, quits HyperCard, goes back to the home stack, or explicitly chooses to finish the session using the monitor configuration.

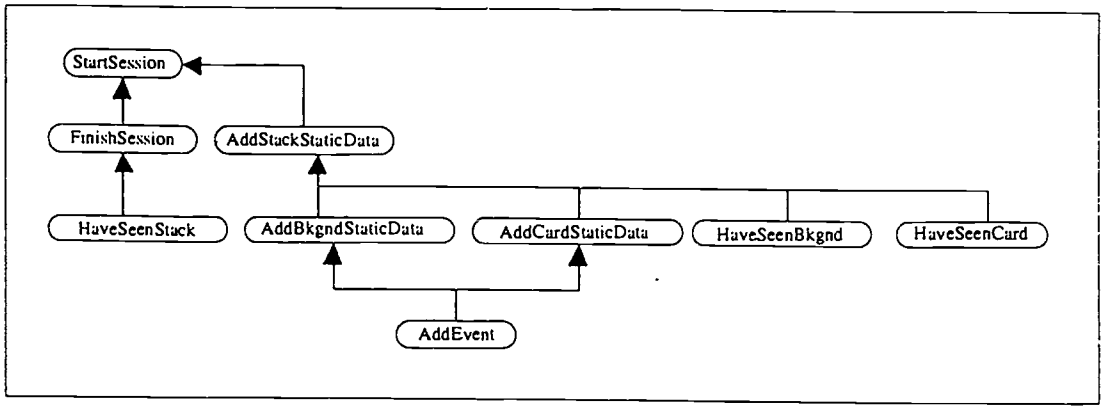


Figure 3. How the different XCMDs rely on each other

HyperCard can be extended through the user of external code modules called XCMDs (pronounced 'x-commands') (Goodman, 1990). The XCMDs are usually written in C or Pascal and are useful to implement things HyperCard is lacking. The monitor is internally divided up into two levels: a series of 11 XCMDs to do all the low-level grind work, and a HyperCard stack called "HyperCard Monitor" with associated HyperTalk handlers to capture the data and call the low-level XCMDs. Implementing most of the monitor in external XCMDs helps to keep it portable to other environments and improves the speed of operation considerably.

10 of the 11 XCMDs control access to the runtime and static data structures that are created during a session. The XCMDs are optimised for speed and most operations are independent on the size of the stack. Those that are not, employ caching mechanisms to improve performance.

One important issue is that the XCMDs are called in a clearly defined order. Figure 3 shows the structure of the first 10 XCMDs by name and a matrix showing which XCMDs have to be called first in order to have well-defined results. The 11th XCMD is the ConvertScript XCMD which makes a script fully compatible with the monitor. The monitor has a section which calls the ConvertScript XCMD to convert a specified stack.

Findings

Whether or not the method of recording user activity employed in this system is a useful approach to monitor systems in general is a good question. There were many key features which made the HyperCard environment a very good basis for such a system, and one was the ability to very easily install methods which would catch events as they happened. What was even better was that the events were at a level which were regarded as most useful when monitoring user activity. From this, it is easy to see that a way that a more general monitor system for an OS could be developed is by letting the OS communicate to and from programs in a similarly high level way. Apple has developed a system-level scripting system called AppleScript (Apple: AppleScript, 1993) which plugs in on top of Apple's System 7 OS. AppleScript provides an interface available to all applications which lets applications send high level messages to each other. For instance "move the selected object 3 pixels to the left and 14 down", "get me information on the person called 'Langley'", or "Get the contents of whatever the user selected".

Although the monitor system was implemented using a combination of HyperTalk scripts and XCMDs, many of the ideas and a lot of the code can be directly used in the more general implementation of a system-level monitor system, including concepts like the caching mechanisms, event data encoding, hierarchical data, and static data models. A system where records were kept of activity in each application, along with sub-records which kept track of windows in each application could be one approach. Another key feature an OS-based monitor system would have would be consistency, and clearly defined ways of performing activities. If there are clearly defined ways of doing things and they are done similarly in different situations, then the monitor system will be able to capitalise on this because it will be able to make assumptions. Assumed knowledge will help the system filter through irrelevant data and derive useful information from minimal data.

The idea of dynamic and static data could be applied as well — there are fixed attributes applications and to a lesser extent windows which can be stored once. In general, the content of application windows is

more flexible than the contents of a card or background, and so perhaps another approach to storing the information in a window would need to be developed. Apple has developed a database of 'things that different types of applications can be told to do' which has thousands of entries, in a hierarchical structure. When people want to extend a message, a class-like approach can be taken so that the message is still identified as a member of the superclass, but is still identifiably different.

Overall, we consider the HyperCard Monitor System to be very effective and many of the general concepts used for the system could easily be extended to OS level if such an OS provided a facility for generating and interpreting high level events of this nature. Performance on a Macintosh Quadra 700 was very good and the resulting data output was very compact — tests revealed about 50k of data per user per $^1/_2$ hr which was several orders of magnitude better than initial prototypes.

One thing that was accepted when doing a monitor system in HyperCard is that there will be constraints and limitations imposed upon what, how and where the monitor can record data, and also to what level of detail and how quickly. There are several areas which could be investigated further to improve the performance and usefulness of the monitor system. Although HyperCard is inherently slow, the monitor's overhead is quite low due to the use of XCMDs and could be improved even more. Most of the speed improvements could come from more clearly defined states, storing less data, storing data in a more compact form, and reducing disk access through caching. A costly process is writing the static data for a card to disk. Often people browse through cards in a stack quickly and so a memory cache of recently used card data could be very helpful.

Another time-consuming process is verifying whether or not a stack has already been seen. Currently this is an O(n) comparison of strings which can in themselves be quite long. This is done every time any of the XCMDs are called. If we could find another way of identifying stacks (perhaps with some simple HyperTalk code keeping information attached to a stack) then performance would improve that way too. In-depth analysis of the monitor's output data could be used to ascertain what data turns out to be the most useful, thus discarding any relatively useless data and keeping the most relevant data more compactly.

A constraint enforced on the current version of the monitor system is that the monitor does not work well with stacks whose structure is altered. Specifically, cards, fields, or buttons added, deleted or sorted are unlikely to work because the runtime information is defined statically whenever a new stack is opened. To add support for addition and deletion of cards, fields, and buttons are non-trivial because of the implications of what a change in the ordering of the stack will do to the references in the data which has already been written to disk.

Some other miscellaneous and all together reasonable constraints are that the disk the monitor is on must be writeable and have enough disk space to accommodate the number of sessions intended to be run on it.

References

Apple Computer, Inc. (1988). The Macintosh HyperCard User's Guide,

Apple Computer, Inc. (1993). Inside Macintosh: AppleScript Language Guide,

Apple Computer, Inc. (1993). Inside Macintosh: Interapplication Communication,

Goodman, D. (1990). The Complete HyperCard XCMD Handbook for HyperCard 2.0 (3rd Edition).

Maurer, H., Sammer, P. & Schneider, A. (1994). Multimedia Systems for the General Public: Experiences at World Expositions and Lessons Learned, *Proceedings of 2nd Interactive Multimedia Symposium, Perth, Australia,* 333-441.

Maurer, H., Schneider, A. (1994). New Aspects of a Hypermedia University Representation, *Proceedings of ED-MEDIA '94, Vancouver,* Canada